A Methodology for Verification and Analysis of Parallel and Distributed Systems Requirement Specifications **

Khalid Lateef Hany Ammar Vinay Mogulothu Tooraj Nikzadeh
Department of Electrical and Computer Engineering
West Virginia University
P. O. Box 6104, Morgantown, WV 26506-6104
lateef@intermetrics.com, ammar@cemr.wvu.edu, vinay@cemr.wvu.edu, tooraj@imake.com

Abstract

Performing rigorous analysis of Parallel and Distributed Systems (PDS) specifications is one of the important tasks during the early stages of development. The ambiguities and errors left unchecked during the analysis phase can creep into design and development phases, resulting in cost and schedule overruns and a less reliable end product. COTS (commercial-off-the-shelf) CASE (Computer Aided Software Engineering) tools can play an important role in the analysis and design phases. However, techniques must be developed to address the shortcomings of CASE tools. A set of such techniques is presented in this paper.

CASE tools can be used to gather PDS specifications in the form of analysis models. The techniques presented in this paper deal with the problem of performing rigorous analysis of PDS specifications originally developed using a CASE tool. The approach is based on integrating a CASE tool with a verification tool based on Coloured Petri Nets (CPNs). CPNs can be used to model and analyze concurrency in specifications and design phases. Dynamic simulations of CPN models can be used to conduct performance/performability analysis as well as risk assessment studies.

1 Introduction

The objective of this work is to develop methods and techniques for generating verification and analysis models from notations used for PDS specifications. These models can be used by the analysts to detect potential problems and prevent these problems from becoming part of the design.

** This work is supported in part by a grant from NASA Goddard to West Virginia University under Contract No. 93-131.

This paper presents a methodology to integrate a CASE environment based on SART (Structured Analysis with Real Time) notation and CPN based verification environment. Semantics mapping rules are used to map SART objects to corresponding CPN objects. The mapping rules presented greatly simplify (in contrast with previously published work [17], [22]) the development of large CPN models. Therefore making these techniques applicable to software models of large distributed systems. Using the CDIF (Case Data Interchange Format) standard, SART models are exported to a Semantics Transfer Utility. This utility maps the SART model semantics to CPN notation. The methodology has been implemented using a COTS CASE tool and Design/CPN environment. A model of a large industrial scale system based on requirement specifications of NASA (National Air and Space Administration, USA) EOS (Earth Observing System) was developed to illustrate the scalability of this methodology. Due to the lack of space, the details of the EOS model and results of the dynamic analysis will be presented in a separate paper.

Background

Requirements supplications languages (or conceptual grammars for requirements specifications) are classified by Fraser and Kumar [12] into two major groups: formal specifications and informal specifications. Informal specifications models supported by CASE tools used in industry are based on SART models or Object-Oriented Analysis models. Formal specifications are based on formal languages such as VDM, Z and Petri Nets.

Informal specification languages use a combination of graphics and semiformal textual grammars to describe and specify software system requirements [3], [4], [12]. These languages are ideal for a developer's environment, as they make it convenient for both user and developer to communicate with each other and refine the user-

description to a set of informal requirements documents. These languages tend to be imprecise and ambiguous. Hence there is a need to use formal specification languages for the requirements analysts domain [20]. A formal notation can be analyzed and manipulated using mathematical operators. Mathematical proof procedures can be used to test and verify the internal consistency and syntactic correctness of the specifications [12]. Formal languages provide exactness and the ability to reason [4]. If the problem can be specified mathematically, then a program can be developed and proven to satisfy the specification.

The CPN modeling environment can be used for software requirements and design specifications. It is especially useful in rigorous analysis of the dynamic behavioral properties such as concurrency analysis, performance analysis, safety, reliability analysis and reachability analysis. Reachability analysis [9] is based on Hierarchical Reachability Graph (HRG). This work shows the applicability of CPN based analysis to large scale models.

Implementation

For many years developers have been using informal techniques such as SART for requirements modeling and specifications. Maier [1] has listed various advantages of using SART methodology. Integrated development environments, e.g. Integrated CASE (ICASE) tools, have evolved to support a number of notations for requirements modeling using SART as well as object-oriented analysis. Such informal specifications are scalable and are being used in large industrial projects. A large gap exists between complex notations used for formal specifications such as CPN and the informal notations used in ICASE tools.

This paper addresses the problem of integrating verification and analysis tools based on CPN with ICASE specification tools as shown in Figure 1. Semantics mapping rules are used in the integration process. The process maps hierarchical requirements models developed in SART notation to hierarchical models in CPN notation. The above approach is implemented using the following tools:

- Teamwork for SART models,
- Design/CPN for CPN models. [1]

1. Teamwork is a COTS CASE tool by Cayanne Software. More information is available at www.cayennesoft.com/ products. At present Design/CPN it is available from University of Aarhus (http://www.daimi.aau.dk/designCPN/).

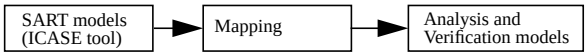

FIGURE 1: The process of mapping SART models to CPN models.

Organization of this paper

The paper is organized as follows. Previous work in related areas is described in Section 2. The layers of tool integration are discussed in Section 4. Data transfer model is described in Section 5. A brief description of semantics mapping rules are presented in the Section 6. Section 7 describes verification and analysis tasks based on dynamic analysis.

2 Previous Work

The literature [2], [12], [24], [1], [21], [5], [14], [17], [22] contains a large body of work on software system modeling and analysis based on mapping informal specifications to formal languages and models. Examples from four different approaches are presented in this section. The last paragraph of this section explains some aspects of CPN and their use in PDS specifications. The four approaches described are summarized as follows:

- SCCS-VP (Value Passing Synchronous Calculus of Communications Systems)
- VDM (Vienna Development Method)
- Approaches used by Mair, Kung, Wieringa and Miriyala
- HLTPN (High Level Timed Petri Nets)

SCCS-VP

This formal notation has been utilized by Hooker and Lockyer [2]. Their method integrates Ward-Mellor SART notation and the SCCS-VP notation. The process of integration involves three major steps. In the first step SART models are used as an input to the SF (Semantic Function) implementation. The output of SF is an SCCP-VP program. In the second step, the SCCS-VP program is translated to a basic SCCS program. In the third step a CWB (Concurrency Workbench) is used for the model checking and simulation of SCCS programs. The Hooker et. al have mentioned limitations in this approach. First there are a lot of manual sub-steps involved. Secondly the tools used are not COTS. The third is the existence of no tool to analyze and simulate SCCS-VP programs. That is why SCCS-VP programs have to be translated to a basic SCCS before CWB can simulate it.

Another example of the use of CCS (Calculus of Communicating System) is the work by Krishnan [8]. He discussed the possibilities and limitations of using CCS

for the purpose of software system analysis. He showed how a formal description can be generated from existing informal documents. The disadvantage, as he pointed out, was that only small, restricted sub-systems can be analyzed using this approach.

VDM

Fraser and Kumar [12] presented VDM [24] based formal models of software systems which can be generated from informal specifications. Informal specifications are expressed in SART. Two methods are discussed in the paper. The first is a cognitive approach which initially uses SART specifications as an aid to human understanding and cognition. This understanding helps the requirements engineer to develop top-level specifications of the systems using the formal VDM specifications language. The second approach is automatic generation of VDM specifications from SART models. Some of the problems with this approach are:

- Discrete signals get mapped to continuous variables
- No mechanism to add timing information
- Uses flattened DFD (Data Flow Diagram) sets

Assuming a VDM based model is available, a process of translating VDM specification into ABC[1] programs is described by Kans et al. [13]. Miranda or Prolog have often been used to prototype the specifications written in the formal notation of VDM. The problem with Miranda and Prolog is that they do not have destructive assignment commands [13]. That makes it difficult to model VDM state changes. Imperative languages like C and Pascal allow state changes to be modelled naturally but lack the expressive power to make prototyping feasible. Kans and Hayton then suggest that ABC should be used for prototyping as it is a simple (yet very powerful) imperative language having expressive power suitable for prototyping.

Approaches used by Mair, Kung, Wieringa and Miriyala

Maier has proposed a technique to link existing methods including real-time structured analysis and design, metrics, performance modeling and quality function deployment [1]. This technique combines four major views for complex system development:

- functional
- physical
- performance and
- management.

An earlier example of modeling software systems is presented in a paper by Kung [21]. This is a graphical approach which can model both static and dynamic aspects of the application in one model. Kung's method provides executable specifications which were translated to a Prolog program to simulate the behavior of the system. This approach combines various modeling techniques such as Entity-Relationship Diagrams, Data flow diagram, Petrinets, Prolog and Relational Calculus. Given the variety of modeling techniques used, it is a complicated approach [12].

Wieringa [5] discusses a concept called Transaction Decomposition Table (TDT). TDT allows the analyst to represent the connection between the static and dynamic system structure.

An approach based on larch shared language is discussed by Miriyala et al. [14]. It is used in a tool called SPECIFIER. The technique uses problem-solving methods such as Schemas, Analogy and Difference-Based Reasoning.

HLTPNs

In [17] and [22] Pezze, Elmstrom, and Lintulampi presented semantics mapping rules to generate HLTPN models from the Ward and Mellor SART notations. The Ward and Mellor SART notations were later modified by Hatley and Pirbhai, and is currently supported by most ICASE tools. The approach in their work is to generate Petrinets which are not to be visible to the analyst. Their objective is to formalize the SART notation and produce an environment supporting the execution of heterogeneous models where parts of the model are implemented in C code. The drawback of this approach is the complexity of the HLTPN models obtained. The mapping rules used in this approach, produce a much more complicated HLTPN model from a relatively simple SART model. While HLTPN models have been proposed to integrate the functional and time aspects in a semantically precise way, we believe Design/CPN adopts a simple time model added to the color Petri nets formalism. HLTPN have been proposed to model the detailed real-time properties of systems, and hence they are more suitable to be used at the detailed design or code level rather than at the requirements analysis level.

1. ABC is a programming language that has been designed and implemented at CWI, the Center for Mathematics and Computer Science, in Amsterdam. It is an imperative language designed originally as a replacement of BASIC. Kans et al., provide the mapping of VDM sets to ABC lists.

3 The Method Presented in This Paper

The focus of our approach is to generate scalable CPN models from the Hatley and Pirbhai SART notations which can be used, refined, and parametrized (for dynamic analysis by the analyst). These models start at a comparable level of abstraction and complexity as the original SART models (with an almost one-to-one mapping of SART objects to CPN objects) and hence they can also be scaled to model large PDS components. The CPN models can be used for reachability and deadlock analysis, performance analysis, performability analysis, reliability analysis, and in general can be used to verify the information system properties as reported in a number of publications [6], [7], [10], [9], [16], [17], [18], [19], [22]. Some of the verifiable properties using Petrinets [10] are:

- derivability and consistent definition of outputs
- performability of processes
- application dependent properties of a concurrent system

In the following section, integration levels among development and verification tools are described.

4 Layers of Tool Integration

Integration of tools can be designed at five levels [29]:

- Carrier Level: Tool integration is accomplished by passing byte streams.
- Lexical Level: At this level tools share data formats and operating conventions that make them interact meaningfully.
- Syntactic Level: The tools agree on a set of data structures and on the rules governing their formation.
- Semantic Level: The tools agree on the structure's semantics which provide enough information to automate development and analysis tasks.
- Method Level: At this level the tools agree on specific process step (e.g. steps of development process such as prototyping, requirements analysis and dynamic modeling).

The approach in this paper is based on the Semantics level and the Method level of tool integration. In the Semantic level of integration, tools agree on the data-structure definitions, as well as the meanings of operations on those structures. At this level there is enough information available to automate development, analysis and design tasks (e.g. code generation tools). This level of agreement between the tools can be achieved by the following methods.

- Hard coding the definition of the data structure and operation specifications into the tools, or
- Including information about the data structures and operations that make up the infrastructure's data repository in the repository itself. Tools can then query such meta-data.

A common definition of the structures' semantics augments the syntactic level information. Most of the integrated tool suites use the first method (hard coding). As the specifications are defined before the tools are written, tool writers know what structures are available, how they are named, the meaning of each structure, and the effect of each operation. A data transfer model based on semantic level integration is described in the following section.

5 The Data Transfer Model

The Data Transfer Model is built on the standards for tool integration and framework for tool-to-tool data transfer. These issues are discussed in the following paragraphs.

5.1 Standards for Tool Integration

There are four standards for tool integration discussed in this section: Semantics Transfer Language, Case Data Interchange Format, Information Resource Dictionary System and Portable Common Tool Environment.

Semantic Transfer Language (STL)

IEEE tool integration standard 1175 or Semantic Transfer Language (STL) has been proposed. This is for non-graphical communications among CASE tools. STL can represent a set of CASE tool information with text and generalized graphical information. It attempts to add more semantics in the information to be exchanged by different tools. It is used as an intermediary language to express a powerful set of tool-generated information among different tools in a conceptualized non-graphical form.

Case Data Interchange Format (CDIF)

CDIF is defined by the Electronic Industries Association as a standard for CASE tool and repository communication. The salient features of this standard are:

- This is used for exchanging information among CASE tools and repositories.
- It includes descriptions, placement and details of text and graphical elements.
- This syntactic/semantics level integration is supported by most CASE tool vendors.

CDIF is also a model of how systems should be built [30]. The basis of CDIF is agreed-on syntactic definitions that let tools exchange the data [29].

Information Resource Dictionary System (IRDS)

An example of a syntactic level integration standard is the IRDS. This is an Entity-Relationship based model that describes the way information is logically stored in the repositories and the methods to be used by tools to access the information. It is a data-base schema which is good for tightly-coupled integration architectures. IRDS defines what should be contained in a repository but leaves its technical design and implementation methods undefined.

Portable Common Tool Environment (PCTE)

PCTE provides a broad and complex set of interface calls (similar to operating-system calls) to underlying facilities that support CASE tools. But it has no explicit functions to support software engineering [29]. It permits the use of CASE tools across operating systems and local area networks. PCTE was developed under the European Strategic Program for Research in Information Technology (ESPRIT). It is supported by several European vendors. It includes a common object oriented layer for transparent information exchange between different types of tools [30]. It is primarily used for C, C++ and Ada environments.

5.2 A Framework for Tool-to-Tool Data Transfer

A framework for tool-to-tool data transfer is shown in Figure 2. This is a general case showing the transfer of data from a CASE tool to a verification tool. The analysis and design data is normally available from the CASE tool database. Most CASE tools provide an access utility to allow others tools to retrieve this data.

Software Bus Facility (SBF) acts as a communication channel between different tools in an integrated environment. Once a data package is available via the SBF, any tool (including the verification tools) can receive this data through a local gateway. Once the local gateway completes the process of receiving data, the Semantics Transfer Utility (STU) converts the input data semantics to data objects of the verification tool. This general framework for tool-to-tool integration is implemented using Teamwork and Design/CPN.

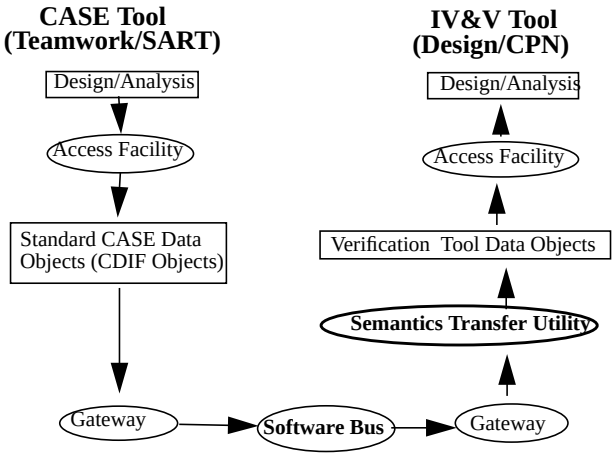

FIGURE 2: General framework for Tool-to-Tool Integration.

6 Semantics mapping rules

The STU uses a set of mapping rules for translating SART objects to the CPN objects. A brief overview of the SART and the CPN environment is given in Section 6.1. In Section 6.2, the semantics mapping rules for translating a SART model to a CPN model are briefly described.

6.1 Description of the SART and the CPN Environments

The following paragraphs describe the SART and the CPN environments.

SART Environment

The SART model components are shown in Figure 3: Data Flow Diagrams (DFD)[1], Control Specifications (C-Spec), Process Specifications (P-Spec) and Data Dictionary Entries (DDE). The DFD at the highest level of abstraction is known as the Context Diagram. This diagram allows three types of objects: a bubble, terminators, data and control flows. The single bubble represents the whole system. Terminators represent the external entities which send or receive data or control signals from the system. The data and control flows connect the terminators and the bubble.

The bubble on the context diagram is decomposed into more bubbles or processes on DFD 0. Each of these

1. Control Flow Diagrams are merged with DFDs

bubbles will be numbered as 1, 2, etc. The analyst determines if any of these processes are primitive.

For a primitive process, a P-Spec is defined. If a process is not primitive, a lower level DFD is used to define it further. For example, in Figure 3, two processes in DFD 0 are defined by the lower level DFD 1 and DFD 2. These steps are repeated until the analyst reaches the primitive level for every process in the model. Definitions for data flows, control flows and stores constitute the data dictionary for a given model. The fields of the data dictionary corresponding to individual flows or stores are called DDE.

Also, a data flow diagram may contain a C-Spec. C-Specs are used to define process activation or handling control flows. A vertical bar on the DFD represents a C-Spec. The C-Spec is further defined on a separate sheet in the SART model. Several representations are in use for defining C-Specs. Some of the examples are State Transition Diagram, Process Activation Table and Decision Table. In this work, only State Transition Diagrams are considered.

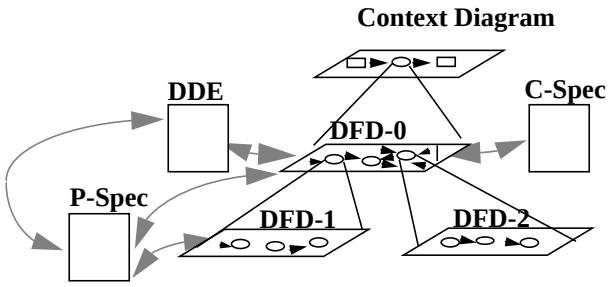

FIGURE 3: Components of SART and their relationship with each other.

CPN Environment

The CPN modeling environment is also hierarchical. A CPN model is arranged in the form of pages as shown in Figure 4. Each page in this case has an associated SART object which is shown just outside the CPN-Page's oval representation. The CPN-PAGE-1 is mapped from the Context Diagram. Similarly CPN-PAGE-2 is mapped from the DFD 0. Other DFD levels are mapped to corresponding CPN pages as shown in the Figure 4.

The oval for Declaration page is not connected in the hierarchy diagram. The reason is that it does not contain a CPN. The definitions of the colors and declaration of CPN variables of different colors are specified in a declaration page. The entries in the declaration page are derived from the DDEs of the SART model.

Every CPN page contains a colored petrinet. A colored petrinet is a petrinet in which different places can have different types of tokens (colors). CPN uses data types, data objects and variables. CPN data types are called colorsets and CPN data objects are called tokens. A CPN consists of the following elements:

- Places (represented by circles): locations for holding data
- Transitions (represented by rectangles): activities that transform data
- Arcs (represented by arrows): connect places with transitions. Arrowhead specifies token flow. Input arcs bring tokens to the transitions and output arcs show the paths leaving a transition.
- Arc inscriptions: Input arc inscriptions specify the data that must exist for an activity to occur, and output arc inscriptions specify the data that will be deposited if an activity occurs. Time stamps on the arc inscriptions represent the delay in the flow of tokens.
- Guards (attribute of transition): define conditions that must be true for an activity to occur. Guards can contain time stamps.
- Code segments (attribute of transition): contain code to implement exact transformation from input tokens to output tokens. The code in these segments is written using CPN Meta Language.

As shown in Figure 4, CPN-PAGE-1 is a superpage for the rest of pages named CPN-PAGE-* (where * is an integer 2 through 5). Inversely CPN-PAGE-2 is a subpage for CPN-PAGE-1.

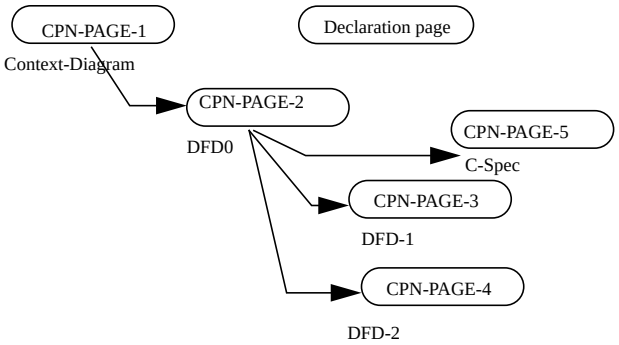

FIGURE 4: CPN page showing model hierarchy

At any given time, the distribution of tokens on places defines the current state of the modeled system. Transitions are connected to a set of input as well as a set of output places. A change in the system state occurs when a transition fires. The firing of a transition constitutes the

removal of tokens from input places and depositing tokens in the output places. Each of these objects (i.e. places, arcs and transitions) have their own sets of attributes. Objects other than the ones just discussed, which may exist on a CPN pages are text blocks and local declaration pages. These are not discussed here for the sake of brevity. Mapping of individual CPN pages from the corresponding DFDs is explained in the following section.

6.2 Mapping SART Objects to CPN Objects

An SART model, as defined in Case Data Interchange Format, is expressed as N-tuple. In this paper only 4-tuple variant of CDIF representation are considered.

- Obj_dfd: A data flow diagram object.
- Obj_dd: Represents the collection of Data Dictionary Entries (DDE). The DDEs hold information on data flows, control flows and data stores.
- obj_std: Used for State Transition Diagram.
- obj_ps: Object containing P-Spec.

The hierarchy definitions are embedded in the individual objects like data flow diagrams, state transition diagrams, data dictionary entries and P-Specs. The mapping procedure presented here uses this description to maintain the hierarchy in the resulting CPN model structure. This procedure was implemented using flex and bison along with mapping rules written in the C-language.

The hierarchical colored petri nets (HCPN) are defined as a tuple:

- S: a finite set of pages such that each page $s \in S$ is a non-hierarchical CPN=($\sum_s$, P_s, T_s, A_s, N_s, C_s, G_s, E_s, I_s), Where:

  $\sum$: finite set of non-empty types (or color sets)

  P: finite set of places

  T: finite set of transitions

  A: finite set of arcs

  N: node function

  C: Color function (P into $\sum$)

  G: guard function. It is defined from T into expressions.

  E: arc expression function. It is defined from A into expressions.

  I: initialization function. It is defined from P into closed expressions.

  Note: Further details are available from the reference [25].

- $SN \subseteq T$: a set of substitution nodes
- SA : a page assignment function. It is defined from SN into S such that no page is a subpage of itself.
- $PN \subseteq P$: a set of port nodes
- PT is port type function. It is defined from PN into {in, out, in/out, general}
- PA: port assignment function. It is defined from SN into binary relations as:

  1. Socket nodes are related to port nodes.

  2. Socket nodes are of the correct type.

  3. Related nodes have identical color sets and equivalent initialization expression.

- $FS \subseteq P_s$ is finite set of fusion sets such that members of a fusion set have identical color sets and equivalent initialization expressions.
- FT is fusion type functions. It is defined from fusion sets into {global, page, instance}.
- $PP \subseteq S_{MS}$ is a multiset of prime pages

The SART objects are mapped to HCPN using the rules given in the following paragraph.

Mapping the SART model to a HCPN model

Rule: $\forall$ obj_dfd $\longrightarrow$ non-hierarchical CPN page

Rule: $\forall$ obj_std $\longrightarrow$ non-hierarchical CPN page

Rule: $\forall$ obj_dd $\longrightarrow$ CPN declaration page

The obj_ps (the process specifications) is not mapped automatically using a semantic transfer utility, rather the analyst converts the process specifications to a code suitable for the CPN. This code can be used in the CPN related functions and the transition code segments.

Mapping the Data Flow Diagram to CPN page

A data flow diagram is a six tuple DFD = (dfd_buble, dfd_store, dfd_term, dfd_tb, dfd_csc, dfd_flow). Each of these objects are further defined in the CDIF standard.

Each CPN page is tuple object ($\sum$, T, P, G, A). These objects are mapped from the data flow diagram object as given below:

Rule: $\forall$ dfd_buble $\longrightarrow$ t such that $t \in T$, and dfd_buble is a primitive one.

Rule: $\forall$ non-primitive dfd_bubble, dfd_buble $\longrightarrow$ SN. Such that the input and output flows of this dfd_buble are mapped to the sockets. The inputs and output flows get mapped to ports in the subpage. In the same step, based on the flow directions on the DFD, PT is also defined from the set of its values {in, out, in/out, general}.

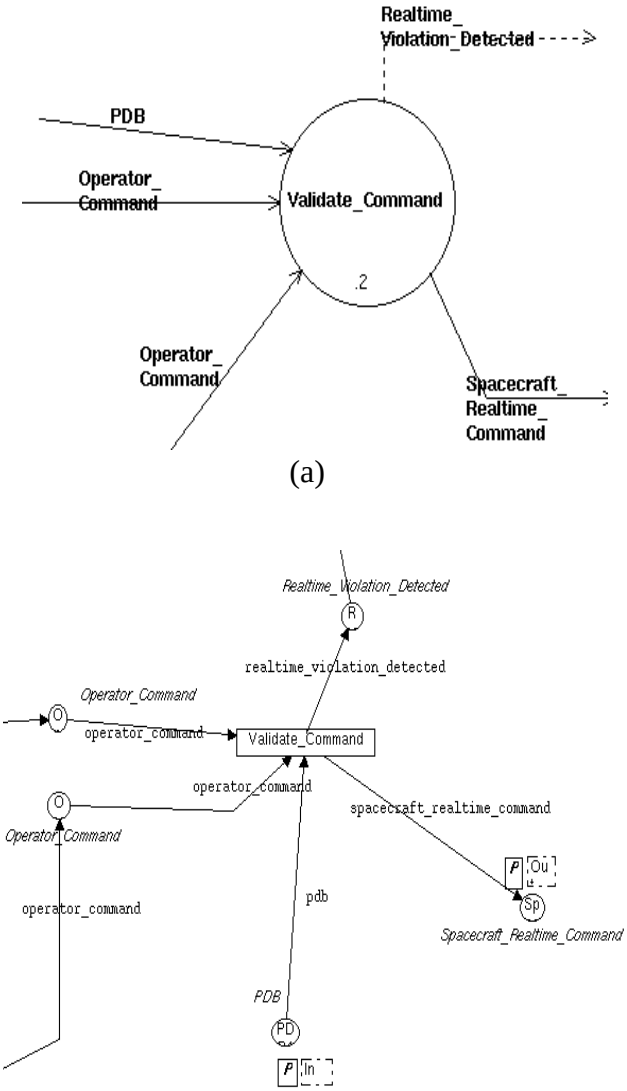

FIGURE 5: (a) SART representation of "Validate_Command", (b) The CPN representation of "Validate_Command".

Rule: $\forall$ dfd_store $\longrightarrow$ f(T,P,G,A)

A dfd_store can be read-only or read-write. The f(T,P,G,A) represents this accordingly.

Rule: $\forall$ dfd_term $\longrightarrow$ P

Rule: $\forall$ dfd_csc $\longrightarrow$ CPN page

Rule: $\forall$ dfd_flow $\longrightarrow$ P

This is just an overview of the mapping rules used for the translation of a data flow diagram.

As shown in Figure 5, a primitive dfd_buble "Validate_Command" (Figure 5-a) is mapped to transition with the same name (Figure 5-b). The input data flows called PDB, Operator_Command_1 and Operator_Command_2 are mapped to the places with the same name. The direction of arcs in Figure 5-b is also mapped from the corresponding direction of data_flows shown in Figure 5-a. The same thing is true for the outputs of the process "Validate_Command". As shown in Figure 5-b, the output places spacecraft_ realtime_ command and PDB are ports. In other words, these places are connected to the corresponding sockets on a superpage. The place spcecraft_realtime_command is an output port. Whereas the place PDB is an input port.

Mapping the Data Dictionary Entries

The data dictionary entries are mapped to corresponding color entries in the global declaration page. In SART the data dictionary element is a three tuple (dde_name, dde_attr_list, edif_body). During the mapping process a dde_name is used for adding a particular color. The objects dde_attr_list (the attribute list for a DDE) and edif_body are used to generate multisets corresponding to a color set in the CPN environment. Table 1 shows some rules for the DDE mapping process.

Table 1: Mapping rules for generating the Global Page of the CPN model.

DDE definition	CPN translation	Remarks
Activity_Violation_Detected = ["TRUE" | "FALSE"]	color Activity_Violation_Detected = **with** TRUE | FALSE;	complete definition
ADCs-Data2 = Information_Dialog + Dialog + Algorithms + Ancillary_Data + Data_Products + Data_Information	color ADCs-Data2 = **record** Information_-Dialog * Dialog *Algorithms *Ancillary_Data * Data_Products * Data_Information;	complete definition
Alarms_Notification = *not-defined*	color Alarms_Notification = **with** Alarms_Notification;	Default color
When the definition of Alert_signal is not found in the DDE table	color Alert_Signal = **with** Alert_Signal	Default color

Mapping the Control Specifications

The C-Spec in a DFD appears as a substitution transition on the corresponding CPN page. A subpage for this substitution transition represents the mapping of its C-

Spec. A state transition diagram is the tuple (std_state, std_tb, std_trans):

- std_state: is the object representing state in the STD
- std_tb: is used to give a definition to an STD as mealy or moore.
- std_trans: represents the transition object.

The mapping rules used are:

Rule: $\forall$std_state $\longrightarrow$ P

Rule: $\forall$std_trans $\longrightarrow$ T

Rule: $\forall$std_tb $\longrightarrow$ f(G,C,P) where both guard (G) as well as code segments (C) are derived from the conditions needed for the change in the system state. The places (P) correspond to the input and output signals.

A cruise control example is presented in Figure 6 to illustrate the mapping process for State Transition Diagrams.

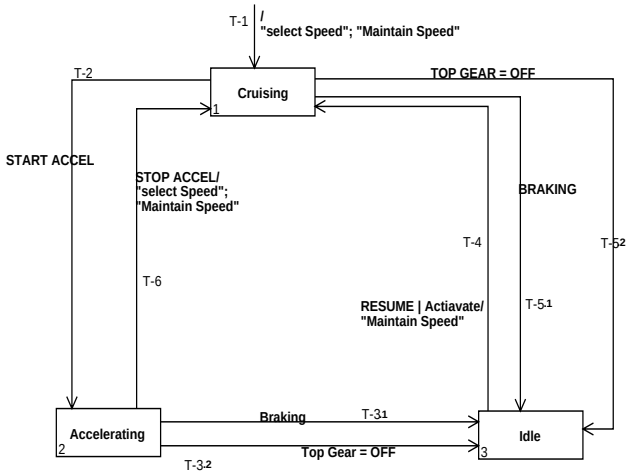

FIGURE 6: Cruise Control STD.

The transitions are marked as T-1 through T-6 for a comparison with transitions in the CPN page shown in Figure 7. The places are named according to the mapping rules given earlier. Two places named as Enable_Select_Speed and Enable_Maintain_Speed are used for enabling/disabling the processes, Select_Speed and Maintain_Speed, respectively. The presence or absence of a token in these places, is used to enable or disable the corresponding process.

Different shades are used to distinguish between places in Figure 7. The places shaded black are the enabling, disabling places. The larger size places shaded grey are the states of the controller. The places with light grey shade are input signals to the controller. There are always tokens present in the input places. For example, the place Braking has a token with a value TRUE if the brakes are on, otherwise it is false. Guards for the transitions are shown in a separate labeled box to reduce diagram clutter.

Note: Arc inscriptions and intermediate place names in Figure 7 are hidden from the view for the sake of clarity.

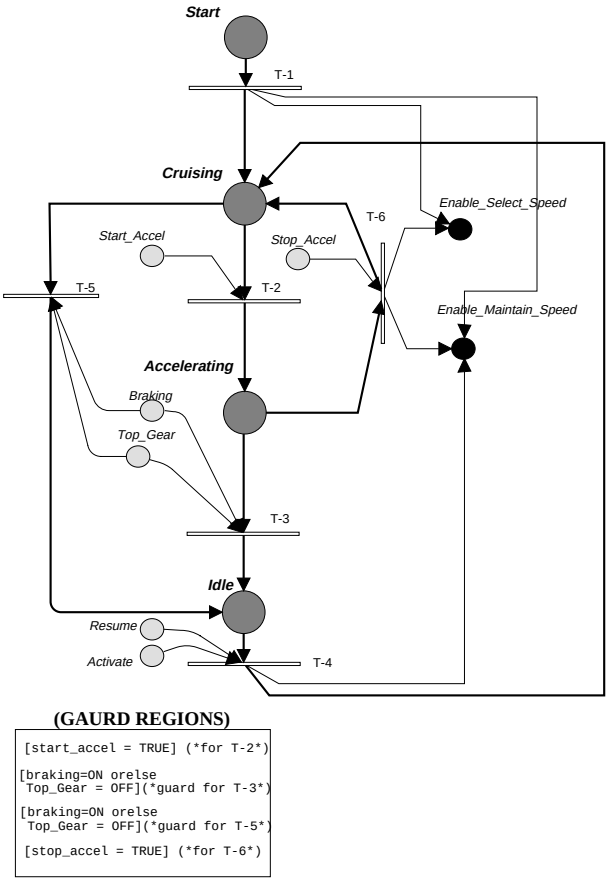

FIGURE 7: CPN diagram for Cruise Control

Once the output file is generated based on these rules, the analyst checks for consistency and completeness. The important aspect related to the enable and disable function of an STD is based on the Hatley and Pirbhai notation. Actions (enabling and disabling) are associated with transitions which are transient in nature. The actions are assumed to continue in effect until the next transition occurs. This means a process activated by a particular action remains activated continuously and continues to respond to changing data inputs until the next transition occurs [26]. The approach used in this paper is slightly different. The enabling places get one token in the state

where a process is to be enabled. Once the token is consumed by the enabled process, there is no more enabling token until the system goes back to the same state. The Hatley Pirbhai approach can be incorporated through slight modifications to our mapping rules.

7 Verification and Analysis Tasks

The methodology described in the previous section can be used to develop dynamic models on which several verification and analysis tasks such as performance analysis and risk assessment can be conducted. A detailed discussion on these tasks is available in the technical report [15]. In this section we briefly describe two analysis techniques conducted using CPN models.

7.1 Performance and Reliability Analysis

The CPN model captures both the static and the dynamic behavior of the specification. In the early design stages the functional modules are relatively large and the knowledge of their execution behavior may be imprecise. As the design progresses and the modules are further resolved, the estimates of their behavior and execution resource characterization become more precise. A CPN model helps in giving the definition and subsequently show dynamic behavior of different components.

System execution scenarios providing the definitions of the external inputs to the model were developed for each simulation run. These simulations were used to verify the dynamic behavior of the original SART specifications. Simulations of the system were also conducted to analyze the performance and performability requirements. The detail about scenarios and the simulation results will be presented in a separate paper.

7.2 Criticality Assessment

For the assessment of criticality, system requirements are classified to determine their relative importance in terms of such factors as performance, mission, safety, complexity, and cost risk. The measure of criticality assessment can be termed as Criticality Factor (CF) [23]. Once the process of assessment is complete, IV&V resources can be concentrated where they are most needed. This also helps while developing test plans to slant the testing toward the more critical requirements. Criticality is based on risk and complexity.

Risks are evaluated by determining the adequacy of the automation decisions and of the software/hardware interfaces on the requirements level. Secondly the risks are evaluated by judgmental evaluation of the type and degree of potential for described categories of failure contingencies. The failure mode effects analysis (FMEA) approach becomes necessary to provide additional depth to the quantification [27]. Analysis of the effects that failures, due to software errors, can have on the system often requires an in-depth study using detailed models of the specifications as presented in this paper. Thus, FMEA and criticality assessments work together to provide a way to best utilize the available V&V resources.

Complexity is related to the cumulative nature of implementing multi-disciplinary software requirements. Complex systems contain interfaces with many subsystems at the same time. Therefore it is necessary to measure the degree to which a given software requirement may impact the baseline system performance requirements if problems occur during program executions. This measure reflects the concept that criticality is directly proportional to the number of connectivities to different subsystems, and that certain subsystem disciplines potentially affect the system performance more than do other disciplines. Complexity analysis can be carried out on a functional basis using the models presented in the previous section to determine the criticality factors of these functions.

8 Conclusions

This paper presented a methodology for generating formal specification models based on CPN. The models are generated from specifications developed in SART. One of the important characteristic of this methodology is scalability. It is adaptable to large scale systems. This can be achieved by mapping the specifications of the model components using a bottom-up approach.

One of the lessons learned in this work is the amount of effort needed to design and implement the semantics mapping utility. The process of mapping a large model also requires the support of a specialized tool to extract the components of a large model from one environment and assemble them in the target environment. This tool is currently being developed as part of an on-going research project at West Virginia University.

The SART specifications used in this paper have been used in many industrial projects and have become a standard notations supported by most CASE tools. In contrast, a large number of notations and techniques for object oriented specifications has been proposed in the literature. Further work is needed to generalize the methodology presented in this paper to use meta-modeling concepts and techniques [31] to accommodate specifications based on the various notations of object oriented models.

9 References

[1] Maier M.W., “Integrated Modeling: A unified Approach to system Engineering”, The Journal of systems and software: Vol 32, pp101-119, 1996.

[2] Hooker s., Lockyer M.A., Fencott P.C., “CASE Support for Methods Integration: Implementation of translation from a structured to a formal notation”, Proceedings of the Methods Integration Workshop, Leeds, 25-26 March 1996, Springer-Verlog.

[3] Amoroso E. G., “Creating formal specifications from informal requirement documents”, ACM SIGSOFT, Software Engineering Notes, Vol 20 no 1, pp 67-70, Jan 1995.

[4] Cooke D., et al., “Languages for the specification of the software”, The Journal of systems and software: Vol 32, pp269-308, 1996.

[5] Wieringa R. J., “Combining Static and Dynamic Modelling Methods: A comparison of Four Methods”, The Computer Journal, Vol 38, No. 1, 1995.

[6] Bucci G., Vicario E., “Compositional Validation of Time-Critical Systems using Communicating Time Petri Nets”, IEEE transactions on software engineering, vol 21 no 12, pp 969-992, December 1995.

[7] Felder M., Mandrioli D., Morzenti A., “Proving properties of real-time systems through logical specifications and petri net models”, IEEE transactions on software engineering, vol 20 no 2, pp 127-141, Feb 1994.

[8] Krishnan P., “Formal methods and design extraction: a pilot study” Information and software technology, vol 36(11), pp 675-681, 1994.

[9] Murata T., Notomi M., “Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis”, IEEE transactions on Software Engineering, pp 325-336, Vol. 20, No. 5, May 1994.

[10] Sakthivel S., Moily J. P., “Analytical verification of information system requirements using Petrinets”, Information and software technology, vol 35 no. 2, pp 89-100, February 1993.

[11] W.T. OLLE, et. al., “Information system Methodologies: A Frame work for Understanding”, Addison-Wesley 1988.

[12] Fraser, M.D. & Kumar, K. “Informal to formal requirement specification languages: Bridging the gap”, IEEE transactions on Software Engineering, pp 454-, Vol. 17, No. 5, May 1991.

[13] Kans A., Hayton C., “Translating VDM specifications into ABC programs”, Information and software technology, vol 36 no. 12, pp 699-706, 1994.

[14] Miriyala K., Harandi M.T., “Automatic Derivation of formal software specifications from informal descriptions”, IEEE transactions on Software Engineering, pp 1126-1142, Vol. 17, No. 10, October 1991.

[15] Ammar H., Lateef K., “IV&V System Integration Methods and Tools”, Technical Report CERC-TR-RN-95-016, November 1995.

[16] Buy U., Sloan R. H., “Analysis of Real-time Programs with simple time petri nets”, ACM software engineering notes, special issue on ISSTA, pp 228-239, 1994.

[17] Pezze M., Elmstrom R., Lintulampi R., “Giving Semantics to SA/RT by means of High-Level timed petri nets”, The international journal of time critical computing systems, Vol. 5, no 2/3, May 1993

[18] Ghezzi et al., “High-level timed petri nets as kernel for executable specifications”, The international journal of time critical computing systems, Vol. 5, no 2/3, May 1993.

[19] Berthomieu et al., “Modeling and verification of time dependent systems using time petri nets”, IEEE transactions on Software Engineering, pp 259-273, Vol. 17, March 1991

[20] N. Dershowitz, “Program abstraction and instantiation”, ACM Trans. Program. Languages and Syst., pp 446-477, Vol. 7, No. 3, October 1985.

[21] Kung C.H., “Conceptual modeling in the context of software development”, IEEE transactions on Software Engineering, pp 1176-1187, Vol. 15, No. 10, 1989

[22] “Automatic translation of SA/RT to high level time Petri nets” Espirit report, IPTES-PDM-17-V2.3. 1994

[23] Robert O. Lewis, “Independent Verification and Validation, A life cycle Engineering Process for Quality Software”, John Wiley and Sons, INC., 1992

[24] Andrews D., “Specification aspects of VDM”, Information and software technology, vol 30, no. 3, pp164-176, April 1988.

[25] Jensen K., “Coloured Petri nets: basic concepts, analysis methods and practical use”, Springer-Verlag, Berlin; New York April 1992.

[26] Hatley Derek J, Pirbhai Imtiaz A, “Strategies for real-time system specification”, Dorset House Pub, New York, NY1987

[27] “Structured development for real-time systems” by Paul T. Ward, Stephen J. Mellor. PUBLISHED: New York, N.Y.: Yourdon Press, 1985

[28] Wikstrom A., “Functional Programming using Standard ML”, Prentice Hall, 1987

[29] Brown, Alan., Mcdermid, John., “Learning from IPSE’s Mistakes”, IEEE Software, March 1992, pp 23-28.

[30] Semich, J.W., “Open CASE Emerges as AD/Cycle Lags”, Datamation, March 1992, pp 30-38.

[31] Nissen H., et al “Managing Multiple Requirements Perspectives with Meta-models,” IEEE Software, Vol. 13, No. 2, March, 1996